

LLR-Based Successive Cancellation List Decoding of Polar Codes

Alexios Balatsoukas-Stimming, *Student Member, IEEE*, Mani Bastani Parizi, *Student Member, IEEE*,
and Andreas Burg, *Member, IEEE*

Abstract—We show that successive cancellation list decoding can be formulated exclusively using log-likelihood ratios. In addition to numerical stability, the log-likelihood ratio based formulation has useful properties which simplify the sorting step involved in successive cancellation list decoding. We propose a hardware architecture of the successive cancellation list decoder in the log-likelihood ratio domain which, compared to a log-likelihood domain implementation, requires less irregular and smaller memories. This simplification together with the gains in the metric sorter, lead to 56% to 137% higher throughput per unit area than other recently proposed architectures. We then evaluate the empirical performance of the CRC-aided successive cancellation list decoder at different list sizes using different CRCs and conclude that it is important to adapt the CRC length to the list size in order to achieve the best error-rate performance of concatenated polar codes. Finally, we synthesize conventional successive cancellation decoders at large block-lengths with the same block-error probability as our proposed CRC-aided successive cancellation list decoders to demonstrate that, while our decoders have slightly lower throughput and larger area, they have a significantly smaller decoding latency.

Index Terms—Successive Cancellation List Decoder, CRC-Aided Successive Cancellation List Decoder, Successive Cancellation Decoder, Polar Codes, Hardware Implementation

I. INTRODUCTION

IN his seminal work [1], Arikan constructed the first class of error correcting codes that can achieve the capacity of any symmetric binary-input discrete memoryless channel (B-DMC) with efficient encoding and decoding algorithms based on *channel polarization*. In particular, Arikan proposed a low-complexity successive cancellation (SC) decoder and proved that the block-error probability of *polar codes* under SC decoding vanishes as their block-length increases. The SC decoder is attractive from an implementation perspective due to its highly structured nature. Several hardware architectures for SC decoding of polar codes have recently been presented in the literature [2]–[8], the first SC decoder ASIC was presented in [9], and simplifications of Arikan’s original SC decoding algorithm are studied in [10]–[13].

A. Balatsoukas-Stimming and A. Burg are with the Telecommunications Circuits Laboratory (TCL), EPFL. Their research is supported by the Swiss National Science Foundation grant 200021_149447.

M. Bastani Parizi is with the Information Theory Laboratory (LTHI), EPFL. His research is supported by the Swiss National Science Foundation grant 200020_146832.

This work has been published in parts in the 39th International Conference on Acoustics, Speech and Signal Processing (ICASSP’2014).

The authors would like to thank Professor Emre Telatar, Professor Ido Tal, Jun Lin, and Bo Yuan for helpful discussions.

Even though the block-error probability of polar codes under SC decoding decays roughly like $O(2^{-\sqrt{N}})$ as a function of the block-length N [14], they do not perform well at low-to-moderate block-lengths. This is to a certain extent due to the sub-optimality of the SC decoding algorithm. To partially compensate for this sub-optimality, Tal and Vardy proposed the successive cancellation list (SCL) decoder whose computational complexity is shown to scale identically to the SC decoder with respect to the block-length [15].

SCL decoding not only improves the block-error probability of polar codes, but also enables one to use *modified polar codes* [16], [17] which are constructed by concatenating a polar code with a cyclic redundancy check (CRC) code as an outer code. Adding the CRC increases neither the computational complexity of the encoder nor that of the decoder by a notable amount, while reducing the block-error probability significantly, making the error-rate performance of the modified polar codes under SCL decoding comparable to the state-of-the-art LDPC codes [16]. In [18] an adaptive variant of the CRC-aided SCL decoder is proposed in order to further improve the block-error probability of modified polar codes while maintaining the average decoding complexity at a moderate level.

The SCL decoding algorithm in [15] is described in terms of likelihoods. Unfortunately, computations with likelihoods are numerically unstable as they are prone to underflows. In recent hardware implementations of the SCL decoder [19]–[23] the stability problem was solved by using log-likelihoods (LLs). However, the use of LLs creates other important problems, such as an irregular memory with varying number of bits per word, as well as large processing elements, making these decoders still inefficient in terms of area and throughput.

Contributions and Paper Outline

After a background review of polar codes and SCL decoding in Section II, in Section III we prove that the SCL decoding algorithm can be formulated exclusively in the *log-likelihood ratio* (LLR) domain, thus enabling area-efficient and numerically stable implementation of SCL decoding. We discuss our SCL decoder hardware architecture in Section IV and leverage some useful properties of the LLR-based formulation in order to *prune* the radix-2L sorter (implementing the sorting step of SCL decoding) used in [19], [24] by avoiding unnecessary comparisons in Section V. Next, in Section VI we see that the LLR-based implementation leads to a significant reduction of the size of our previous hardware architecture [19], as

well as to an increase of its maximum operating frequency. We also compare our decoder with the recent SCL decoder architectures of [22], [23] and show that our decoder can have more than 100% higher throughput per unit area than those architectures.

Besides the implementation gains, it is noteworthy that most processing blocks in practical receivers process the data in the form of LLRs. Therefore, the LLR-based SCL decoder can readily be incorporated into existing systems while the LL-based decoders would require extra processing stages to convert the channel LLRs into LLs. In fairness, we note that one particular advantage of LL-based SCL decoders is that the algorithmic simplifications of [10]–[13] can readily be applied to the SCL decoder [25], while in order to apply those simplifications to an LLR-based SCL decoder one has to rely on *approximations* [26].

Finally, we show that a CRC-aided SCL decoder can be implemented by incorporating a CRC unit into our decoder, with almost no additional hardware cost, in order to achieve significantly lower block-error probabilities. As we will see, for a fixed information rate, the choice of CRC length is critical in the design of the modified polar code to be decoded by a CRC-aided SCL decoder. In Section VI-E we provide simulation results showing that for small list sizes a short CRC will improve the performance of SCL decoder while larger CRCs will even degrade its performance compared to a standard polar code. As the list size gets larger, one can increase the length of CRC in order to achieve considerably lower block-error probabilities.

An interesting question, which is, to the best of our knowledge, still unaddressed in the literature, is whether it is better to use SC decoding with long polar codes or SCL decoding with short polar codes. In Section VIII we study two examples of long polar codes that have the same block-error probability under SC decoding as our (1024, 512) modified polar codes under CRC-aided SCL decoding and compare the synthesis results of the corresponding decoders.

II. BACKGROUND

Notation: Throughout this paper, boldface letters denote vectors. The elements of a vector \mathbf{x} are denoted by x_i and \mathbf{x}_l^m means the sub-vector $[x_l, x_{l+1}, \dots, x_m]^T$ if $m \geq l$ and the null vector otherwise. If $\mathcal{I} = \{i_1, i_2, \dots\}$ is an ordered set of indices, $\mathbf{x}_{\mathcal{I}}$ denotes the sub-vector $[x_{i_1}, x_{i_2}, \dots]^T$. For a positive integer m , $\llbracket m \rrbracket \triangleq \{0, 1, \dots, m-1\}$. If \mathcal{S} is a countable set, $|\mathcal{S}|$ denotes its cardinality. $\log(\cdot)$ and $\ln(\cdot)$ denote base-2 and natural logarithm respectively. We follow the standard coding theory notation and denote a code of block-length N and rate $\frac{K}{N}$ as an “ (N, K) code.”

For $N = 2^n$, $n \geq 1$, let \mathbf{U} be a uniformly distributed random vector in $\{0, 1\}^N$ and suppose the random vector $\mathbf{X} \in \{0, 1\}^N$ is computed from \mathbf{U} through the linear transform

$$\mathbf{X} = \mathbf{G}_n \mathbf{U}, \quad \text{where} \quad \mathbf{G}_n \triangleq \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{\otimes n} \mathbf{B}_n, \quad (1)$$

where $\otimes n$ denotes the n th Kronecker power of the matrix and

\mathbf{B}_n is the bit-reversal permutation.¹

If \mathbf{X} is transmitted via N independent uses of the B-DMC $W : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X} = \{0, 1\}$ is the input alphabet and $W(y|x)$ is the probability distribution function of the output letter $Y \in \mathcal{Y}$ when x is transmitted, the conditional distribution of the output vector $\mathbf{Y} \in \mathcal{Y}^N$ is

$$W^N(\mathbf{y}|\mathbf{x}) \triangleq \Pr[\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}] = \prod_{i=0}^{N-1} W(y_i|x_i), \quad (2)$$

for $\forall \mathbf{x} \in \mathcal{X}^N$ and $\mathbf{y} \in \mathcal{Y}^N$. Equivalently, the distribution of \mathbf{Y} conditioned on $\{\mathbf{U} = \mathbf{u}\}$ is

$$W_n(\mathbf{y}|\mathbf{u}) \triangleq \Pr[\mathbf{Y} = \mathbf{y}|\mathbf{U} = \mathbf{u}] = W^N(\mathbf{y}|\mathbf{G}_n \mathbf{u}), \quad (3)$$

for $\forall \mathbf{u} \in \mathcal{X}^N$ and $\forall \mathbf{y} \in \mathcal{Y}^N$ with $W^N(\mathbf{y}|\mathbf{x})$ as in (2).²

‘Synthesize’ N B-DMCs, $W_n^{(i)}$, $i \in \llbracket N \rrbracket$ by defining $W_n^{(i)}$ as the B-DMC whose input is U_i and whose output is the vector of physical channel outputs \mathbf{Y} together with all preceding elements of \mathbf{U} , \mathbf{U}_0^{i-1} as side information, considering all following elements of \mathbf{U} as i.i.d. Bernoulli noise. Thus, the transition probabilities of $W_n^{(i)} : \mathcal{X} \rightarrow \mathcal{Y} \times \mathcal{X}^i$ are

$$W_n^{(i)}(\mathbf{y}, \mathbf{u}_0^{i-1}|u_i) \triangleq \sum_{\mathbf{u}_{i+1}^{N-1} \in \mathcal{X}^{N-i-1}} \frac{1}{2^{N-1}} W_n(\mathbf{y}|\mathbf{u}). \quad (4)$$

Arıkan shows that as $n \rightarrow \infty$, these synthetic channels *polarize* to ‘easy-to-use’ B-DMCs [1, Theorem 1]. That is, all except a vanishing fraction of them will be either almost-noiseless channels (whose output is almost a deterministic function of the input) or useless channels (whose output is almost statistically independent of the input). Furthermore, the fraction of almost-noiseless channels is equal to the symmetric capacity of the underlying channel—the highest rate at which reliable communication is possible through W when the input letters $\{0, 1\}$ are used with equal frequency [27].

A. Polar Codes and Successive Cancellation Decoding

Having transformed N identical copies of a ‘moderate’ B-DMC W into N ‘extremal’ B-DMCs $W_n^{(i)}$, $i \in \llbracket N \rrbracket$, Arıkan constructs capacity-achieving *polar codes* by exploiting the almost-noiseless channels to communicate information bits.

1) *Polar Coding:* In order to construct a polar code of rate R and block length N for a channel W , the indices of the NR least noisy synthetic channels $W_n^{(i)}$, $i \in \llbracket N \rrbracket$ are selected as the *information* indices denoted by $\mathcal{A} \subset \llbracket N \rrbracket$. The sub-vector $\mathbf{u}_{\mathcal{A}}$ will be set to the NR data bits to be sent to the receiver and $\mathbf{u}_{\mathcal{F}}$, where $\mathcal{F} = \llbracket N \rrbracket \setminus \mathcal{A}$, is fixed to some *frozen* vector which is known to the receiver. The vector \mathbf{u} is then encoded to the codeword \mathbf{x} through (1) using $O(N \log N)$ binary additions (cf. [1, Section VII]) and transmitted via N independent uses of the channel W .

The receiver observes the channel output vector \mathbf{y} and estimates the elements of the $\mathbf{u}_{\mathcal{A}}$ *successively* as follows: Suppose

¹Let \mathbf{v} and \mathbf{u} be two length $N = 2^n$ vectors and index their elements using binary sequences of length n , $(b_1, b_2, \dots, b_n) \in \{0, 1\}^n$. Then $\mathbf{v} = \mathbf{B}_n \mathbf{u}$ iff $v_{(b_1, b_2, \dots, b_n)} = u_{(b_n, b_{n-1}, \dots, b_1)}$ for $\forall (b_1, b_2, \dots, b_n) \in \{0, 1\}^n$.

²Following the convention in probability theory, we denote the realizations of the random vectors \mathbf{U} , \mathbf{X} , and \mathbf{Y} as \mathbf{u} , \mathbf{x} , and \mathbf{y} respectively.

the information indices are ordered as $\mathcal{A} = \{i_1, i_2, \dots, i_{NR}\}$ (where $i_j < i_{j+1}$). Having the channel output, the receiver has all the required information to decode the input of the synthetic channel $W_n^{(i_1)}$ as \hat{u}_{i_1} , as, in particular, $\mathbf{u}_0^{i_1-1}$ is a part of the known sub-vector $\mathbf{u}_{\mathcal{F}}$. Since this synthetic channel is assumed to be almost-noiseless by construction, $\hat{u}_{i_1} = u_{i_1}$ with high probability. Subsequently, the decoder can proceed to index i_2 as the information required for decoding the input of $W_n^{(i_2)}$ is now available. Once again, this estimation is with high probability error-free. As detailed in Algorithm 1, this process is continued until all the information bits have been estimated.

Algorithm 1: SC Decoding [1].

```

1 for  $i = 0, 1, \dots, N-1$  do
2   if  $i \notin \mathcal{A}$  then                                // frozen bits
3      $\hat{u}_i \leftarrow u_i$ ;
4   else                                              // information bits
5      $\hat{u}_i \leftarrow \arg \max_{u_i \in \{0,1\}} W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | u_i)$ ;
6 return  $\hat{\mathbf{u}}_{\mathcal{A}}$ ;

```

2) *SC Decoding as a Greedy Tree Search Algorithm:* Let

$$\mathcal{U}(\mathbf{u}_{\mathcal{F}}) \triangleq \{\mathbf{v} \in \mathcal{X}^N : \mathbf{v}_{\mathcal{F}} = \mathbf{u}_{\mathcal{F}}\} \quad (5)$$

denote the set of 2^{NR} possible length- N vectors that the transmitter can send. The elements of $\mathcal{U}(\mathbf{u}_{\mathcal{F}})$ are in one-to-one correspondence with 2^{NR} leaves of a binary tree of height N : the leaves are constrained to be reached from the root by following the direction u_i at all levels $i \in \mathcal{F}$. Therefore, any decoding procedure is essentially equivalent to picking a *path* from the root to one of these leaves on the binary tree.

In particular, an optimal ML decoder, associates each path with its likelihood (or any other *path metric* which is a monotone function of the likelihood) and picks the path that maximizes this metric by exploring *all* possible paths:

$$\hat{\mathbf{u}}_{\text{ML}} = \arg \max_{\mathbf{v} \in \mathcal{U}(\mathbf{u}_{\mathcal{F}})} W_n(\mathbf{y} | \mathbf{v}). \quad (6)$$

Clearly such an optimization problem is computationally infeasible as the number of paths, $|\mathcal{U}(\mathbf{u}_{\mathcal{F}})|$, grows exponentially with the block-length N .

The SC decoder, in contrast, finds a sub-optimal solution by maximizing the likelihood via a *greedy* one-time-pass through the tree: starting from the root, at each level $i \in \mathcal{A}$, the decoder extends the existing path by picking the child that maximizes the *partial likelihood* $W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | u_i)$.

3) *Decoding Complexity:* The computational task of the SC decoder is to calculate the pairs of likelihoods $W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | u_i)$, $u_i \in \{0, 1\}$ needed for the decisions in line 5 of Algorithm 1. Since the decisions are binary, it is sufficient to compute the *decision log-likelihood ratios (LLRs)*,

$$\mathbb{L}_n^{(i)} \triangleq \ln \left(\frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | 1)} \right), \quad i \in \llbracket N \rrbracket. \quad (7)$$

It can be shown (see [1, Section VII] and [2]) that the

decision LLRs (7) can be computed via the recursions,

$$\begin{aligned} \mathbb{L}_s^{(2i)} &= f_- \left(\mathbb{L}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathbb{L}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])} \right), \\ \mathbb{L}_s^{(2i+1)} &= f_+ \left(\mathbb{L}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathbb{L}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}, \mathbf{u}_s^{(2i)} \right), \end{aligned}$$

for $s = n, n-1, \dots, 1$, where $f_- : \mathbb{R}^2 \rightarrow \mathbb{R}$ and $f_+ : \mathbb{R}^2 \times \{0, 1\} \rightarrow \mathbb{R}$ are defined as

$$f_-(\alpha, \beta) \triangleq \ln \left(\frac{e^{\alpha+\beta} + 1}{e^{\alpha} + e^{\beta}} \right), \quad (8a)$$

$$f_+(\alpha, \beta, u) \triangleq (-1)^u \alpha + \beta, \quad (8b)$$

respectively. The recursions terminate at $s = 0$ where

$$\mathbb{L}_0^{(i)} \triangleq \ln \left(\frac{W(y_i | 0)}{W(y_i | 1)} \right), \quad \forall i \in \llbracket N \rrbracket,$$

are *channel LLRs*. The *partial sums* $\mathbf{u}_s^{(i)}$ are computed starting from $\mathbf{u}_n^{(i)} \triangleq \hat{u}_i$, $\forall i \in \llbracket N \rrbracket$ and setting

$$\begin{aligned} \mathbf{u}_{s-1}^{(2i-[i \bmod 2^{s-1}])} &= \mathbf{u}_s^{(2i)} \oplus \mathbf{u}_s^{(2i+1)}, \\ \mathbf{u}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])} &= \mathbf{u}_s^{(2i+1)}, \end{aligned}$$

for $s = n, n-1, \dots, 1$.

Therefore, the entire set of $N \log N$ LLRs $\mathbb{L}_s^{(i)}, s \in \{1, \dots, n\}, i \in \llbracket N \rrbracket$ can be computed using $O(N \log N)$ updates since from each pair of LLRs at *stage* s , a pair of LLRs at stage $s+1$ is calculated using f_- and f_+ update rules (see Figure 1). Additionally the decoder must keep track of $N \log N$ partial sums $\mathbf{u}_s^{(i)}, s \in \llbracket n \rrbracket, i \in \llbracket N \rrbracket$ and update them after decoding each bit \hat{u}_i .

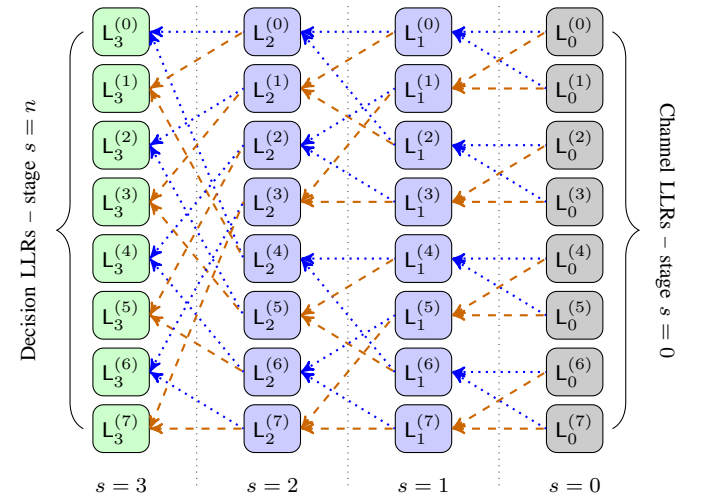


Fig. 1. The butterfly computational structure of the SC decoder for $n = 3$; blue and orange arrows show f_- and f_+ updates respectively.

Remark. It can easily be checked that (cf. [2])

$$f_-(\alpha, \beta) \approx \tilde{f}_-(\alpha, \beta) \triangleq \text{sign}(\alpha) \text{sign}(\beta) \min\{|\alpha|, |\beta|\}, \quad (9)$$

where \tilde{f}_- is a ‘hardware-friendly’ function as it involves only the easy-to-implement $\min\{\cdot, \cdot\}$ operation (compared to f_- which involves exponentiations and logarithms). For a hardware implementation of the SC decoder the update rule f_- is replaced by \tilde{f}_- . Given f_+ , such an approximation is called the “min-sum approximation” of the decoder.

B. Successive Cancellation List Decoding

The *successive cancellation list (SCL)* decoding algorithm, introduced in [15], converts the greedy one-time-pass search of SC decoding into a breadth-first search under a complexity constraint in the following way: At each level $i \in \mathcal{A}$, instead of extending the path in only one direction, the decoder is duplicated in two parallel *decoding threads* continuing in either possible direction. However, in order to avoid the exponential growth of the number of decoding threads, as soon as the number of parallel decoding threads reaches L , at each step $i \in \mathcal{A}$, only L threads corresponding the L most likely paths (out of $2L$ tentatives) are retained.³ The decoder eventually finishes with a *list* of L candidates $\hat{\mathbf{u}}[\ell]$, $\ell \in \llbracket L \rrbracket$, corresponding to L (out of 2^{NR}) paths on the binary tree and declares the most likely of them as the final estimate. This procedure is formalized in Algorithm 2. Simulation results in [15] show that for a (2048, 1024) polar code, a relatively small list size of $L = 32$ is sufficient to have a close-to-ML block-error probability.

Algorithm 2: SC List Decoding [15]

```

1  $\mathcal{L} \leftarrow \{0\}$ ; // start with a single active thread
2 for  $i = 0, 1, \dots, N-1$  do
3   if  $i \notin \mathcal{A}$  then // frozen bits
4      $\hat{u}_i[\ell] \leftarrow u_i$  for  $\forall \ell \in \mathcal{L}$ ;
5   else // information bits
6     if  $|\mathcal{L}| < L$  then // duplicate all the threads
7       foreach  $\ell \in \mathcal{L}$  do
8          $\text{duplicatePath}(\ell)$ ;
9     else
10      Compute  $P_{\ell,u} = W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell] | u)$ , for  $\forall \ell \in \mathcal{L}$ 
        and  $\forall u \in \{0, 1\}$ ;
11       $\tau \leftarrow$  the median of  $2L$  numbers  $P_{\ell,u}$ ;
12      foreach  $\ell \in \mathcal{L}$  such that  $P_{\ell,0} < \tau$  and  $P_{\ell,1} < \tau$  do
13        Kill the thread  $\ell$  and set  $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\ell\}$ ;
14      for  $\ell \in \mathcal{L}$  do
15        if  $P_{\ell,u} > \tau$  while  $P_{\ell,u \oplus 1} < \tau$  then
16           $\hat{u}_i[\ell] \leftarrow u$ ;
17        else // both  $P_{\ell,0}$  and  $P_{\ell,1}$  are  $\geq \tau$ 
18           $\text{duplicatePath}(\ell)$ ;
19  $\ell^* \leftarrow \arg \max_{\ell \in \mathcal{L}} W_n^{(N-1)}(\mathbf{y}, \hat{\mathbf{u}}_0^{N-1}[\ell] | \hat{u}_N[\ell])$ ;
20 return  $\hat{\mathbf{u}}_{\mathcal{A}}[\ell^*]$ ;
21 subroutine  $\text{duplicatePath}(\ell)$ 
22   Copy the thread  $\ell$  into a new thread  $\ell' \notin \mathcal{L}$ ;
23    $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell'\}$ ;
24    $\hat{u}_i[\ell] \leftarrow 0$ ;
25    $\hat{u}_i[\ell'] \leftarrow 1$ ;

```

While a naive implementation of SCL decoder would have a decoding complexity of at least $\Omega(L \cdot N^2)$ (due to $\Theta(L \cdot N)$ duplications of data structures of size $\Omega(N)$ in lines 8 and 18 of Algorithm 2), a clever choice of data structures together with the recursive nature of computations enables the authors of [15] to use a copy-on-write mechanism and implement the decoder in $O(L \cdot N \log N)$ complexity.

³Although it is not necessary, L is normally a power of 2

C. CRC-Aided Successive Cancellation List Decoder

In an extended version of their work [16], Tal and Vardy observe that when the SCL decoder fails, in most of the cases, the correct path (corresponding to $\mathbf{u}_{\mathcal{A}}$) is among the L paths the decoder has ended up with. The decoding error happens since there exists another more likely path which is selected in line 19 of Algorithm 2 (note that in such situations the ML decoder would also fail). They, hence, conclude that the performance of polar codes would be significantly improved if the decoder were assisted for its final choice.

Such an assistance can be realized by adding r more non-frozen bits (i.e., creating a polar code of rate $R + r/N$ instead of rate R) to the underlying polar code and then setting the last r non-frozen bits to an r -bit CRC of the first NR information bits (note that the *effective* information rate of the code is unchanged). The SCL decoder, at line 19, first discards the paths that do not pass the CRC and then chooses the most likely path among the remaining ones. Since the CRC can be computed efficiently [28, Chapter 7], this does not notably increase the computational complexity of the decoder. The empirical results of [16] show that a (2048, 1024) concatenated polar code (with a 16-bit CRC) decoded using a list decoder with list size of $L = 32$, outperforms the existing state-of-the-art WiMAX (2304, 1152) LDPC code [29].

Remark. According to [30], the empirical results of [16] on the CRC-aided successive cancellation list decoder (CA-SCLD) are obtained using a (2048, 1040) (outer) polar code with the last 16 unfrozen bits being the CRC of the first 1024 information bits and the results on the non-CRC aided (standard) SCL decoder are obtained using a (2048, 1024) polar code—both having an effective information rate of $\frac{1}{2}$. In [17], [20], [23] the CA-SCLD is realized by keeping the number of non-frozen bits fixed and setting the last r of them to the CRC of the preceding $NR - r$ information bits. This reduces the effective information rate of the code and makes the comparison between the SCLD and the CA-SCLD unfair.⁴

III. LLR-BASED PATH METRIC COMPUTATION

Algorithms 1 and 2 are both valid high-level descriptions of SC and SCL decoding, respectively. However, for implementing these algorithms, the stability of the computations is crucial. Both algorithms summarized in Section II are described in terms of likelihoods which are *not* safe quantities to work with; a decoder implemented using the likelihoods is prone to underflow errors as they are typically tiny numbers.⁵

Considering the binary tree picture that we provided in Section II-A2, the decision LLRs $L_n^{(i)}(7)$ summarize all the necessary information for choosing the most likely child among two children of the same parent at level i . In Section II-A3 we saw that having this type of decisions in the conventional SC decoder allows us to implement the computations in the LLR domain using numerically stable operations. However, in the SCL decoder, the problem is to choose the L most likely

⁴In [18] this discrepancy is not clarified. However, this work focuses only on CA-SCLD without comparison of the performance of a SCLD to a CA-SCLD.

⁵As noticed in [16], it is not difficult to see that $W_n^{(i)}(\mathbf{y}, \mathbf{u}_0^{i-1} | u_i) \leq 2^{-i}$.

children out of $2L$ children of L different parents (lines 10 to 18 of Algorithm 2). For these comparisons the decision log-likelihood *ratios* $L_n^{(i)}$ alone are not sufficient.

Consequently, the software implementation of the decoder in [15] implements the decoder in the likelihood domain by rewriting the recursions of Section II-A3 for computing pairs of likelihoods $W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}|u_i)$, $u_i \in \{0, 1\}$ from pairs of channel likelihoods $W(y_i|x_i)$, $x_i \in \{0, 1\}$, $i \in \llbracket N \rrbracket$. To avoid underflows, at each intermediate step of the updates the likelihoods are scaled by a common factor such that $P_{\ell,u}$ in line 10 of Algorithm 2 is proportional to $W(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell]|u_i)$ [16].

Alternatively, such a normalization step can be avoided by performing the computations in the log-likelihood (LL) domain, i.e., by computing the pairs $\ln(W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell]|u))$, $u \in \{0, 1\}$ for $i \in \llbracket N \rrbracket$ as a function of channel log-likelihood pairs $\ln(W(y_i|x_i))$, $x_i \in \{0, 1\}$, $i \in \llbracket N \rrbracket$ [19]. Log-likelihoods provide some numerical stability, but still involve some issues compared to the log-likelihood *ratios* as we shall discuss in Section IV.

Luckily, we shall see that the decoding paths can still be ordered according to their likelihoods using all of the past decision LLRs $L_n^{(j)}$, $j \in \{0, 1, \dots, i\}$ and the trajectory of each path as summarized in the following theorem.

Theorem 1. *For each path ℓ and each level $i \in \llbracket N \rrbracket$ let the path-metric be defined as:*

$$\text{PM}_{\ell}^{(i)} \triangleq \sum_{j=0}^i \ln(1 + e^{-(1-2\hat{u}_j[\ell]) \cdot L_n^{(j)}[\ell]}), \quad (10)$$

where

$$L_n^{(i)}[\ell] = \ln \left(\frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell]|1)} \right),$$

is the log-likelihood ratio of bit u_i given the channel output \mathbf{y} and the past trajectory of the path $\hat{\mathbf{u}}_0^{i-1}[\ell]$.

If all the information bits are uniformly distributed in $\{0, 1\}$, for any pair of paths ℓ_1, ℓ_2 ,

$$W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell_1]|\hat{u}_i[\ell_1]) < W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1}[\ell_2]|\hat{u}_i[\ell_2])$$

if and only if

$$\text{PM}_{\ell_1}^{(i)} > \text{PM}_{\ell_2}^{(i)}.$$

In view of Theorem 1, one can implement the SCL decoder using L parallel low-complexity and stable LLR-based SC decoders as the underlying building blocks and, in addition, keep track of L path-metrics. The metrics can be updated successively as the decoder proceeds by setting

$$\text{PM}_{\ell}^{(i)} = \phi(\text{PM}_{\ell}^{(i-1)}, L_n^{(i)}[\ell], \hat{u}_i[\ell]), \quad (11a)$$

where the function $\phi : \mathbb{R}_+^2 \times \{0, 1\} \rightarrow \mathbb{R}_+$ is defined as

$$\phi(\mu, \lambda, u) \triangleq \mu + \ln(1 + e^{-(1-2u)\lambda}). \quad (11b)$$

As shown in Algorithm 3, the paths can be compared based on their likelihood using the values of the associated path metrics.

Algorithm 3: LLR-based formulation of SCL Decoding

```

1  $\mathcal{L} \leftarrow \{0\}$ ; // start with a single active thread
2  $\text{PM}_0^{(0)} \leftarrow 0$ ;
3 for  $i = 0, 1, \dots, N-1$  do
4   Compute  $L_n^{(i)}[\ell]$  for  $\forall \ell \in \mathcal{L}$ ; // parallel SC decoders
5   if  $i \notin \mathcal{A}$  then // frozen bits
6      $(\hat{u}_i[\ell], \text{PM}_{\ell}^{(i)}) \leftarrow (u_i, \phi(\text{PM}_{\ell}^{(i-1)}, L_n^{(i)}[\ell], u_i))$  for
7        $\forall \ell \in \mathcal{L}$ ; // cf. (11b)
8   else // information bits
9     Set  $P_{\ell,u} \leftarrow \phi(\text{PM}_{\ell}^{(i-1)}, L_n^{(i)}, u)$  for  $\forall \ell \in \mathcal{L}$  and
10       $\forall u \in \{0, 1\}$ ; // cf (11b)
11     if  $|\mathcal{L}| < L$  then // duplicate all the threads
12       foreach  $\ell \in \mathcal{L}$  do
13         duplicatePath( $\ell$ );
14     else
15        $\tau \leftarrow$  the median of  $2L$  numbers  $P_{\ell,u}$ ;
16       foreach  $\ell \in \mathcal{L}$  such that  $P_{\ell,0} > \tau$  and  $P_{\ell,1} > \tau$  do
17         Kill the thread  $\ell$  and set  $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\ell\}$ ;
18       for  $\ell \in \mathcal{L}$  do
19         if  $P_{\ell,u} > \tau$  while  $P_{\ell,u \oplus 1} \leq \tau$  then
20            $(\hat{u}_i[\ell], \text{PM}_{\ell}^{(i)}) \leftarrow (u, P_{\ell,u})$ ;
21         else // both  $P_{\ell,0}$  and  $P_{\ell,1}$  are  $\leq \tau$ 
22           duplicatePath( $\ell$ );
23    $\ell^* \leftarrow \arg \min_{\ell \in \mathcal{L}} \text{PM}_{\ell}^{(N)}$ ;
24   return  $\hat{\mathbf{u}}_{\mathcal{A}}[\ell^*]$ ;
25   subroutine duplicatePath( $\ell$ )
26     Copy the thread  $\ell$  into a new thread  $\ell' \notin \mathcal{L}$ ;
27      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell'\}$ ;
28      $(\hat{u}_i[\ell], \text{PM}_{\ell}^{(i)}) \leftarrow (0, P_{\ell,0})$ ;
29      $(\hat{u}_i[\ell'], \text{PM}_{\ell'}^{(i)}) \leftarrow (1, P_{\ell,1})$ ;

```

Before proving Theorem 1 let us provide an intuitive interpretation of our metric. Since

$$\ln(1 + e^x) \approx \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } x \geq 0, \end{cases}$$

the update rule (11) is well-approximated if we replace ϕ with $\tilde{\phi} : \mathbb{R}_+^2 \times \{0, 1\} \rightarrow \mathbb{R}_+$ defined as

$$\tilde{\phi}(\mu, \lambda, u) \triangleq \begin{cases} \mu & \text{if } u = \frac{1}{2}[1 - \text{sign}(\lambda)], \\ \mu + |\lambda| & \text{otherwise.} \end{cases} \quad (12)$$

We also note that $\frac{1}{2}[1 - \text{sign}(L_n^{(i)}[\ell])]$ is the direction that the LLR (given the past trajectory $\hat{\mathbf{u}}_0^{i-1}[\ell]$) suggests. This is the same decision that a SC decoder would have taken if it was to estimate the value of u_i at step i given the past set of decisions $\hat{\mathbf{u}}_0^{i-1}[\ell]$ (cf. line 5 in Algorithm 1). Equation (12) shows that if at step i the ℓ th path does not follow the direction suggested by $L_n^{(i)}[\ell]$ it will be penalized by an amount $\approx |L_n^{(i)}[\ell]|$.

Having such an interpretation, one might immediately conclude that the path that SC decoder would follow will always have the lowest penalty hence is always declared as the output of the SCL decoder. So why should the SCL decoder exhibit a better performance compared to the SC decoder? The answer is that such a reasoning is correct only if *all* the elements of \mathbf{u} are information bits. As soon as the decoder encounters a

frozen bit, the path metric is updated based on the likelihood of that frozen bit, given the past trajectory of the path and the a-priori known value of that bit (cf. line 6 in Algorithm 3). This can penalize the SC path by a considerable amount, if the value of that frozen bit does not agree with the LLR given the past trajectory (which is an indication of a preceding erroneous decision), while keeping some other paths unpenalized.

We devote the rest of this section to the proof of Theorem 1.

Lemma 1. *If U_i is uniformly distributed in $\{0, 1\}$, then,*

$$\frac{W_n^{(i)}(\mathbf{y}, \mathbf{u}_0^{i-1} | u_i)}{\Pr[U_0^i = \mathbf{u}_0^i | \mathbf{Y} = \mathbf{y}]} = 2 \Pr[\mathbf{Y} = \mathbf{y}].$$

Proof: Since $\Pr[U_i = u_i] = \frac{1}{2}$ for $\forall u_i \in \{0, 1\}$,

$$\begin{aligned} \frac{W_n^{(i)}(\mathbf{y}, \mathbf{u}_0^{i-1} | u_i)}{\Pr[U_0^i = \mathbf{u}_0^i | \mathbf{Y} = \mathbf{y}]} &= \frac{\Pr[\mathbf{Y} = \mathbf{y}, U_0^i = \mathbf{u}_0^i]}{\Pr[U_i = u_i] \Pr[U_0^i = \mathbf{u}_0^i | \mathbf{Y} = \mathbf{y}]} \\ &= \frac{\Pr[\mathbf{Y} = \mathbf{y}] \Pr[U_0^i = \mathbf{u}_0^i | \mathbf{Y} = \mathbf{y}]}{\Pr[U_i = u_i] \Pr[U_0^i = \mathbf{u}_0^i | \mathbf{Y} = \mathbf{y}]} = 2 \Pr[\mathbf{Y} = \mathbf{y}]. \quad \blacksquare \end{aligned}$$

Proof of Theorem 1: It is sufficient to show

$$\text{PM}_\ell^{(i)} = -\ln \left(\Pr[U_0^i = \hat{\mathbf{u}}_0^i[\ell] | \mathbf{Y} = \mathbf{y}] \right). \quad (13)$$

Having shown (13), Theorem 1 will follow as an immediate corollary to Lemma 1 (since the channel output \mathbf{y} is fixed for all decoding paths). Since the path index ℓ is fixed on both sides of (10) we will drop it in the sequel. Let

$$\Lambda_n^{(i)} \triangleq \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | 0)}{W_n^{(i)}(\mathbf{y}, \mathbf{u}_0^{i-1} | 1)} = \frac{\Pr[\mathbf{Y} = \mathbf{y}, U_0^{i-1} = \hat{\mathbf{u}}_0^{i-1}, U_i = 0]}{\Pr[\mathbf{Y} = \mathbf{y}, U_0^{i-1} = \hat{\mathbf{u}}_0^{i-1}, U_i = 1]}$$

(the last equality follows since $\Pr[U_i = 0] = \Pr[U_i = 1]$), and observe that showing (13) is equivalent to proving

$$\Pr[U^i = \hat{\mathbf{u}}^i | \mathbf{Y} = \mathbf{y}] = \prod_{j=0}^i (1 + (\Lambda_n^{(j)})^{-(1-2\hat{u}_j)})^{-1}. \quad (14)$$

Since

$$\begin{aligned} \Pr[\mathbf{Y} = \mathbf{y}, U_0^{i-1} = \hat{\mathbf{u}}_0^{i-1}] &= \sum_{\hat{\mathbf{u}}_i \in \{0, 1\}} \Pr[\mathbf{Y} = \mathbf{y}, U_0^i = \hat{\mathbf{u}}_0^i] \\ &= \Pr[\mathbf{Y} = \mathbf{y}, U_0^i = \hat{\mathbf{u}}_0^i] (1 + (\Lambda_n^{(i)})^{-(1-2\hat{u}_i)}), \end{aligned}$$

$$\begin{aligned} \Pr[\mathbf{Y} = \mathbf{y}, U_0^i = \hat{\mathbf{u}}_0^i] &= (1 + (\Lambda_n^{(i)})^{-(1-2\hat{u}_i)})^{-1} \Pr[\mathbf{Y} = \mathbf{y}, U_0^{i-1} = \hat{\mathbf{u}}_0^{i-1}]. \quad (15) \end{aligned}$$

Repeated application of (15) (for $i-1, i-2, \dots, 0$) yields

$$\Pr[\mathbf{Y} = \mathbf{y}, U_0^i = \hat{\mathbf{u}}_0^i] = \prod_{j=0}^i (1 + (\Lambda_n^{(j)})^{-(1-2\hat{u}_j)})^{-1} \Pr[\mathbf{Y} = \mathbf{y}].$$

Dividing both sides by $\Pr[\mathbf{Y} = \mathbf{y}]$ proves (14). \blacksquare

IV. SCL DECODER HARDWARE ARCHITECTURE

In this section, we show how the LLR-based path metric which we derived in the previous section can be exploited in order to derive a very efficient LLR-based SCL decoder hardware architecture. More specifically, we give a detailed description of each unit of our LLR-based SCL decoder architecture, which essentially consists of L parallel SC decoders along with a path management unit which coordinates the tree search. Moreover, we highlight the advantages over our previous LL-based architecture described in [19]. Our SCL decoder consists of five units: the *memories unit*, the *metric computation unit* (MCU), the *metric sorting unit*, the *address translation unit*, and the *control unit*. An overview of the SCL decoder is shown in Figure 2.

A. LLR and Path Metric Quantization

All LLRs are quantized using a Q -bit signed uniform quantizer with step size $\Delta = 1$. The path metrics are unsigned numbers which are quantized using M bits. Since the path metrics are initialized to 0 and, in the worst case, they are incremented by $2^{Q-1} - 1$ for each bit index i , the maximum possible value of a path metric is $N(2^{Q-1} - 1) = 2^{n+Q-1} - 2^n < 2^{n+Q-1}$. Hence, at most $M = n + Q - 1$ bits are sufficient to ensure that there will be no overflows in the path metric. In practice, any path that gets continuously harshly penalized will most likely be discarded. Therefore, as we will see in Section VI, much fewer bits are sufficient in practice for the quantization of the path metrics.

B. Metric Computation Unit

The computation of the LLRs (line 4 of Algorithm 3) can be fully parallelized. Consequently, the MCU consists of L parallel SC decoder cores which implement the SC decoding update rules and compute the L decision LLRs using the semi-parallel SC decoder architecture of [5] with P processing elements (PEs). These decision LLRs are required to update the path metrics $\text{PM}_\ell^{(i)}$. Whenever the L decision LLRs have been computed, the MCUs wait for one clock cycle. During this single clock cycle, the path metrics $\text{PM}_\ell^{(i)}$ are updated and sorted. Moreover, based on the result of metric sorting, the partial sum, path, and pointer memories are also updated in the same clock cycle, as described in the sequel.

Each decoder core reads its input LLRs from one of the L physical LLR memory banks based on an address translation performed by the pointer memory (described in more detail in Section IV-D).

C. Memory Unit

1) *LLR Memory:* The channel LLRs are fixed during the decoding process of a given codeword, meaning that an SCL decoder requires only one copy of the channel LLRs. These are stored in a memory which is $\frac{N}{P}$ words deep and QP bits wide. On the other hand, the internal LLRs of the intermediate stages of the SC decoding (metric computation) process are different for each path $\ell \in \llbracket L \rrbracket$. Hence we require L physical LLR memory banks with $N - 1$ memory positions per bank.

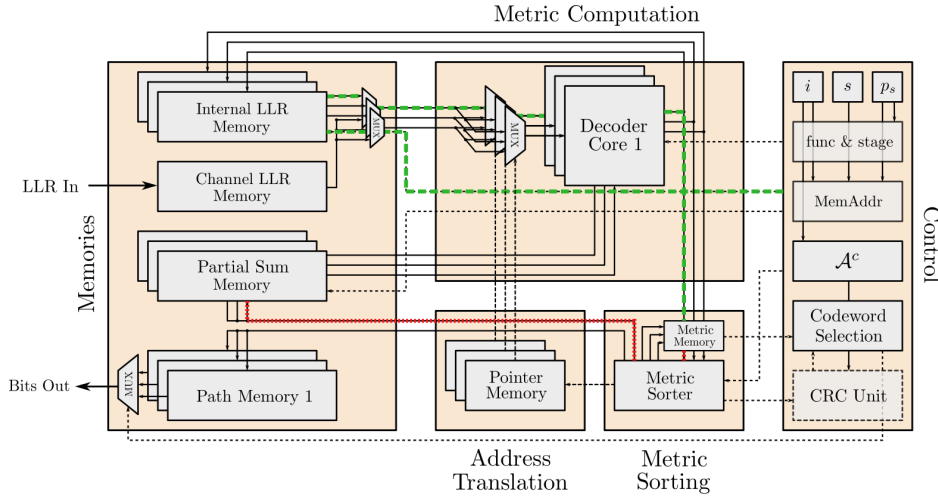


Fig. 2. Overview of the SCL decoder architecture. Details on the i, s, p_s , as well as the func & stage and MemAddr components inside the control unit, which are not described in this paper, can be found in [19]. The dashed green and the dotted red line show the critical paths for $L = 2$ and $L = 4, 8$ respectively.

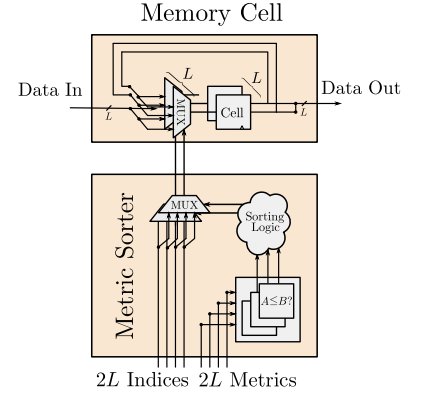


Fig. 3. Bit-cell copying mechanism controlled by the metric sorter.

All LLR memories have two reads ports, so that all P PEs can read their two Q -bit input LLRs simultaneously. Here, register based storage cells are used to implement all the memories.

2) *Path Memory*: The path memory consists of L N -bit registers, denoted by $\hat{u}[\ell]$, $\ell \in \llbracket L \rrbracket$. When a path ℓ needs to be duplicated, the contents of $\hat{u}[\ell]$ are copied to $\hat{u}[\ell']$, where ℓ' corresponds to an inactive path (cf. line 25 of Algorithm 3). The decoder is stalled for one clock cycle in order to perform the required copy operations by means of $N L \times L$ crossbars which connect each $\hat{u}[\ell]$, $\ell \in \llbracket L \rrbracket$ with all other $\hat{u}[\ell']$, $\ell' \in \llbracket L \rrbracket$. The copy mechanism is presented in detail in Figure 3, where we show how each memory bit-cell is controlled based on the results of the metric sorter. After path ℓ has been duplicated, one copy is extended with the bit value $\hat{u}_i[\ell] = 0$, while the other is updated with $\hat{u}_i[\ell'] = 1$ (cf. lines 26 and 27 of Algorithm 3).

3) *Partial Sum Memory*: The partial sum memory consists of L PSNs, where each PSN is implemented as in [5]. When a path $\ell \in \llbracket L \rrbracket$ needs to be duplicated, the contents of the PSN ℓ are copied to another PSN ℓ' , where ℓ' corresponds to an inactive path (cf. line 25 of Algorithm 3). Copying is performed in parallel with the copy of the path memory in a single clock cycle by using $N L \times L$ crossbars which connect each PSN $\ell \in \llbracket L \rrbracket$ with all other PSNs $\ell' \in \llbracket L \rrbracket$. If PSN ℓ was duplicated, one copy is updated with the bit value $\hat{u}_i[\ell] = 0$, while the other copy is updated with $\hat{u}_i[\ell'] = 1$. If a single copy of PSN ℓ was kept, then this copy is updated with the value of $\hat{u}_i[\ell]$ that corresponds to the surviving path.

D. Address Translation Unit

The copy-on-write mechanism used in [15] (which is fully applicable to LLRs) is sufficient to ensure that the decoding complexity is $O(LN \log N)$, but it is not ideal for a hardware implementation as, due to the recursive implementation of the computations, it still requires copying the internal LLRs which is costly in terms of power, decoding latency, and silicon area. On the other hand, a sequential implementation of the

computations enables a more hardware-friendly solution [19], where each path has its own virtual internal LLR memory, the contents of which are physically spread across all of the L LLR memory banks. The translation from virtual memory to physical memory is done using a small *pointer memory*. When a path ℓ needs to be duplicated, as with the partial sum memory, the contents of row ℓ of the pointer memory are copied to some row corresponding to a discarded path through the use of $L \times L$ crossbars.

E. Metric Sorting Unit

The metric sorting unit contains a *path metric memory* and a *path metric sorter*. The path metric memory stores the L path metrics $PM_{\ell}^{(i)}$ using M bits of quantization for each metric. In order to find the median τ at each bit index i (line 13 of Algorithm 3), the path metric sorter sorts the $2L$ candidate path metrics $P_{\ell,u}$, $\ell \in \llbracket L \rrbracket$, $u \in \{0, 1\}$ (line 8 of Algorithm 3). The path metric sorter takes the $2L$ path metrics as an input and produces the sorted path metrics, as well as the path indices ℓ and bit values u which correspond to the sorted path metrics as an output. Since decoding can not continue before the surviving paths have been selected, the metric sorter is a crucial component of the SCL decoder. Hence, we will discuss the sorter architecture in more detail in Section V.

F. Control Unit

The control unit generates all memory read and write addresses as in [5]. Moreover, the control unit contains the codeword selection unit and the optional CRC unit.

The CRC unit contains L r -bit CRC memories, where r is the number of CRC bits. A bit-serial implementation of a CRC computation unit is very efficient in terms of area and path delay, but it requires a large number of clock cycles to produce the checksum. However, this computation delay is masked by the bit-serial nature of the SCL decoder itself and, thus, has no impact on the number of clock cycles required

to decode each codeword. Before decoding each codeword, all CRC memories are initialized to r -bit all-zero vectors. For each $\hat{u}_i[\ell]$, $i \in \mathcal{A}$, the CRC unit is activated to update the CRC values. When decoding finishes, the CRC unit declares which paths $\ell \in \llbracket L \rrbracket$ pass the CRC. When a path is duplicated the corresponding CRC memory is copied by means of $L \times L$ crossbars (like the partial sums and the path memory).

If the CRC unit is present, the codeword selection unit selects the most likely path (i.e., the path with the lowest metric) out of the paths that pass the CRC. Otherwise, the codeword selection unit simply chooses the most likely path.

G. Clock Cycles Per Codeword

Let the total number of cycles required for metric sorting at all information indices $i \in \mathcal{A}$ be denoted by $D_{\text{MS}}(\mathcal{A})$. As we will see in Section V-C, the sorting latency depends on the number of information bits and may depend on the pattern of frozen and information bits as well (both of these parameters can be deduced given \mathcal{A}). Then, our SCL decoder requires

$$D_{\text{SCL}}(N, P, \mathcal{A}) = 2N + \frac{N}{P} \log \frac{N}{4P} + D_{\text{MS}}(\mathcal{A}) \quad (16)$$

cycles to decode each codeword.

H. Advantages Over LL-based SCL Decoder Implementation

The LLs in the SCL decoders of [19]–[23] are all positive numbers and the corresponding LL-domain update rules involve only additions and comparisons. This means that, as decoding progresses through the decoding stages, the dynamic range of the LLs is increased. Thus, in order to avoid catastrophic overflows, all LLs in stage s are quantized using $Q + s$ bits. In the LLR-based implementation of this paper, the LLRs of all stages can be quantized using the same number of bits since the update rules involve both addition and subtraction and the dynamic range of the LLRs in different stages is smaller than that of the LLs. This leads to a regular memory where all elements have the same bit-width. Hence, as we will see in Section VI, using LLRs significantly reduces the total size of the decoder. In addition, the PEs in the LL-based SCL decoder architectures of [19], [20] must support computations with a much larger bit-width than the ones in our LLR-based SCL decoder architecture. Moreover, it turns out that the path metric in the LLR-based decoder can be quantized using much fewer bits than in the LL-based decoder, hence decreasing the delay and the size of the comparators in the metric sorting unit. Finally, the LLR-based formulation enables us to significantly simplify the metric sorter, as explained in the following section.

V. SIMPLIFIED SORTER

For large list sizes ($L \geq 4$), the maximum (critical) delay path passes through the metric sorter, thus reducing the maximum operating frequency of the decoder in [19], [24]. It turns out that the LLR-based path metric we introduced in Theorem 1 has some properties (which the LL-based path metric lacks) that can be used to simplify the sorting task.

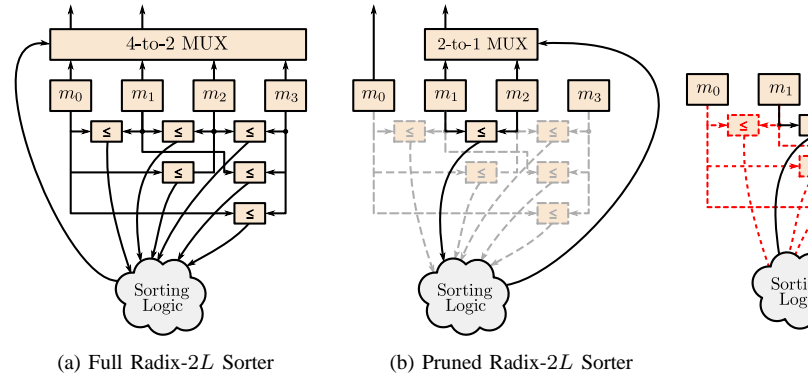


Fig. 4. Radix-2L sorters for $L = 2$

To this end, we note that the $2L$ real numbers that have to be sorted in line 13 of Algorithm 3 are not arbitrary; half of them are the previously existing path-metrics (which can be assumed to be already sorted as a result of decoding the preceding information bit) and the rest are obtained by adding positive real values (the absolute value of the corresponding LLRs) to the existing path metrics. Moreover, we do not need to sort *all* these $2L$ potential path metrics; a sorted list of the L smallest path metrics is sufficient.

Hence, the sorting task of the SCL decoder can be formalized as follows: Given a sorted list of L numbers

$$\mu_0 \leq \mu_1 \leq \dots \leq \mu_{L-1}$$

a list of size $2L$, $\mathbf{m} = [m_0, m_1, \dots, m_{2L-1}]$ is created by setting

$$m_{2\ell} := \mu_\ell \quad \text{and} \quad m_{2\ell+1} := \mu_\ell + a_\ell, \quad \ell \in \llbracket L \rrbracket,$$

where $a_\ell \geq 0$, for $\forall \ell \in \llbracket L \rrbracket$. The problem is to find a sorted list of L smallest elements of \mathbf{m} when the elements of \mathbf{m} have the following two properties: for $\forall \ell \in \{0, 1, \dots, L-2\}$,

$$m_{2\ell} \leq m_{2(\ell+1)}, \quad (17a)$$

$$m_{2\ell} \leq m_{2\ell+1}. \quad (17b)$$

A. Full Radix-2L Sorter

The most straightforward way to solve our problem is to sort the list \mathbf{m} up to the L -th element. This can be done using a simple extension of the radix-2L sorter described in [31], which blindly compares every pair of elements $(m_\ell, m_{\ell'})$ and then combines the results to find the first L smallest elements. This is the solution we used in [19], which requires $\binom{2L}{2} = L(2L-1)$ comparators together with L 2L-to-1 multiplexers (see Figure 4a). The *sorting logic* combines the results of all comparators in order to generate the control signal for the multiplexers (cf. [31] for details). The maximum path delay of the radix-2L sorter is mainly determined by the complexity of the sorting logic, which in turn depends on the number of comparator results that need to be processed.

B. Pruned Radix-2L Sorter

The *pruned radix-2L* sorter presented in this section reduces the complexity of the sorting logic of the radix-2L sorter

and, thus, also the maximum path delay, by eliminating some pairwise comparisons whose results are either already known or irrelevant.

Proposition 1. *It is sufficient to use a pruned radix-2L sorter that involves only $(L-1)^2$ comparators to find the L smallest elements of \mathbf{m} . This sorter is obtained by*

- (a) *removing the comparisons between every even-indexed element of \mathbf{m} and all following elements, and*
- (b) *removing the comparisons between m_{2L-1} and all other elements of \mathbf{m} .*

Proof: Properties (17a) and (17b) imply $m_{2\ell} \leq m_{\ell'}$ for $\forall \ell' > 2\ell$. Hence, the outputs of these comparators are known. Furthermore, as we only need the first L elements of the list sorted and m_{2L-1} is never among the L smallest elements of \mathbf{m} , we can always replace m_{2L-1} by $+\infty$ (pretending the result of the comparisons involving m_{2L-1} is known) without affecting the output of the sorter.

In step (a) we have removed $\sum_{\ell=0}^{L-1} (2L-1-2\ell) = L^2$ comparators and in step (b) $(L-1)$ comparators (note that in the full sorter m_{2L-1} is compared to all $(2L-1)$ preceding elements but L of them correspond to even-indexed elements whose corresponding comparators have already been removed in step (a)). Hence we have $L(2L-1) - L^2 - (L-1) = (L-1)^2$ comparators. ■

Besides the $(L-1)^2$ comparators, the pruned radix-2L sorter requires $L-1$ $(2L-2)$ -to-1 multiplexers (see Figure 4b).

The pruned radix-2L sorter is derived based on the assumption that the existing path metrics are already sorted. This assumption is violated when the decoder reaches the first frozen bit after the first cluster of information bits; at each frozen index, some of the path-metrics are unchanged and some are increased by an amount equal to the absolute value of the LLR. In order for the assumption to hold when the decoder reaches the next cluster of information bits, the L existing path metrics have to be sorted before the decoding of this cluster starts. The existing pruned radix-2L sorter can be used for sorting L arbitrary positive numbers as follows.

Proposition 2. *Let a_0, a_1, \dots, a_{L-1} be L non-negative numbers. Create a list of size $2L$ as*

$$\mathbf{b} \triangleq [0, a_0, 0, a_1, \dots, 0, a_{L-2}, a_{L-1}, +\infty].$$

Feeding this list to the pruned radix-2L sorter will result in an output list of the form

$$[\underbrace{0, 0, \dots, 0}_{L-1 \text{ zeros}}, a_{(0)}, a_{(1)}, \dots, a_{(L-1)}, +\infty]$$

where $a_{(0)} \leq a_{(1)} \leq \dots \leq a_{(L-1)}$ is the ordered permutation of a_0, a_1, \dots, a_{L-1} .

Proof: It is clear that the assumptions (17a) and (17b) hold for \mathbf{b} . The proof of Proposition 1 shows if the last element of the list is additionally known to be the largest element, the pruned radix-2L sorter sorts the entire list. ■

Note that while the same comparator network of a pruned radix-2L sorter is used for sorting L numbers, L separate L -to-1 multiplexers are required to output the sorted list.

C. Latency of Metric Sorting

We assume that the sorting procedure is carried out in a single clock cycle. A decoder based on the full radix-2L sorter, only needs to sort the path metrics for the information indices, hence, the total sorting latency of such an implementation is

$$D_{\text{MS}}(\mathcal{A}) = |\mathcal{A}| = NR \text{ cycles.} \quad (18)$$

Using the pruned radix-2L sorter, additional sorting steps are required at the end of each contiguous set of frozen indices. Let $F_C(\mathcal{A})$ denote the number of *clusters* of frozen bits for a given information set \mathcal{A} .⁶ The metric sorting latency using the pruned radix-2L sorter is then

$$D_{\text{MS}}(\mathcal{A}) = |\mathcal{A}| + F_C(\mathcal{A}) = NR + F_C(\mathcal{A}) \text{ cycles.} \quad (19)$$

VI. IMPLEMENTATION RESULTS

In this section, we present synthesis results for our SCL decoder architecture. For fair comparison with [23], we use a TSMC 90 nm technology with a typical timing library (1 V supply voltage, 25°C operating temperature) and our decoder of [19] is re-synthesized using this technology. All synthesis runs are performed with timing constraints that are not achievable, in order to assess the maximum achievable operating frequency of each design, as reported by the synthesis tool. For our synthesis results, we have used $P = 64$ PEs per SC decoder core, as in [5], [19]. The hardware *efficiency* is defined as the throughput per unit area and it is measured in Mbps/mm². The decoding throughput of all decoders is:

$$T_{\text{SCL}}(N, P, \mathcal{A}, f) = \frac{f \cdot N}{D_{\text{SCL}}(N, P, \mathcal{A})}, \quad (20)$$

where f is the operating frequency of the decoder.

We first compare the LLR-based decoder of this work with our previous LL-based decoder [19], in order to demonstrate the improvements obtained by moving to an LLR-based formulation of SCL decoding. Then, we examine the effect of using the pruned radix-2L sorter on our LLR-based SCL decoder. Finally, we compare our LLR-based decoder with the LL-based decoder of [23] (since [23] is an improved version of [20], we do not compare directly with [20]) and [22]. A direct comparison with the SCL decoders of [21], [26] is unfortunately not possible, as the authors do not report their synthesis results in terms of mm². Finally, we provide some discussion on the effectiveness of a CA-SCLD.

A. Quantization Parameters

In Figure 5, we present the FER of floating-point and fixed-point implementations of an LL-based and an LLR-based SCL decoder for a (1024, 512) polar code as a function of SNR.⁷ For the floating-point simulations we have used the exact implementation of the decoder, i.e., for computing the LLRs

⁶ More precisely we assume $\mathcal{F} = \bigcup_{j=1}^{F_C(\mathcal{A})} \mathcal{F}_j$ such that (i) $\mathcal{F}_j \cap \mathcal{F}_{j'} = \emptyset$ if $j \neq j'$, i.e., $\{\mathcal{F}_j : j = 1, \dots, F_C(\mathcal{A})\}$ is a partition of \mathcal{F} ; (ii) for every j , \mathcal{F}_j is a contiguous subset of $\llbracket N \rrbracket$; and (iii) for every pair $j \neq j'$, $\mathcal{F}_j \cup \mathcal{F}_{j'}$ is not a contiguous subset of $\llbracket N \rrbracket$. It can be easily checked that such a partition always exists and is unique.

⁷ The code is optimized for $E_b/N_0 = 2\text{dB}$ and constructed using the Monte-Carlo method of [1, Section IX].

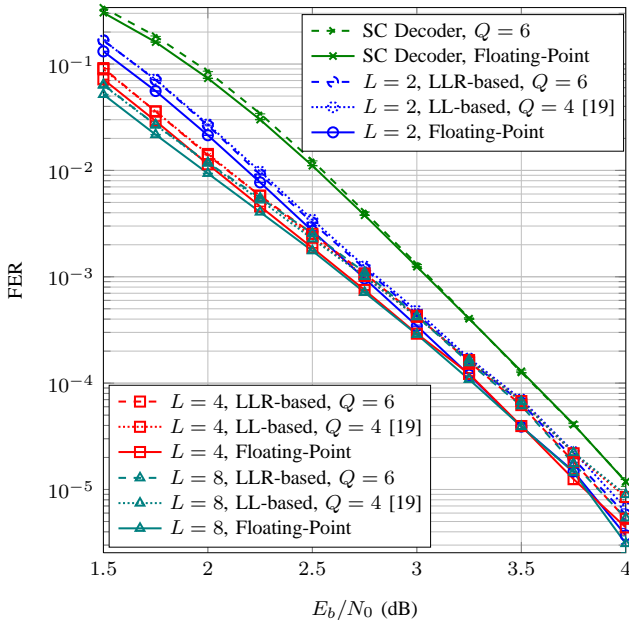


Fig. 5. The performance of floating-point vs. fixed-point SCL decoders. $M = 8$ quantization bits are used for the path metric in fixed-point SCL decoders.

the update rule f_- of (8a) is used and the path metric is iteratively updated according to (11). In contrast, for the fixed-point simulations we have used the min-sum approximation of the decoder (i.e., replaced f_- with \tilde{f}_- as in (9)) and the approximated path metric update rule of (12).

We observe that the LL-based and the LLR-based SCL have practically indistinguishable FER performance when quantizing the channel LLs and the channel LLRs with $Q = 4$ bits and $Q = 6$ bits respectively. Moreover, in our simulations we observe that the performance of the LL and the LLR-based SCL decoder is degraded significantly when $Q < 6$ and $Q < 4$, respectively. As discussed in Section IV-A, metric quantization requires at most $M = n + Q - 1$ bits. However, in practice, much fewer bits turn out to be sufficient. For example, in our simulations for $N = 1024$ and $Q = 6$, setting $M = 8$ leads to the same performance as the worst-case $M = 15$, while setting $M = 7$ results in a significant performance degradation due to metric saturation. Thus, all synthesis results of this section are obtained for $Q = 4$ for the LL-based decoder of [19], and $Q = 6$ and $M = 8$ for the LLR-based decoder for a fair (i.e., iso-FER) comparison.

The authors of [22] do not provide the FER curves for their fixed-point implementation of SCLD and the authors of [23] only provide the FERs for a CA-SCLD [23, Figure 2]. Nevertheless, we assume their quantization schemes will not result in a better FER performance for a standard SCLD than that of [19] since they both implement exactly the same algorithm as in [19] (using a different architecture than [19]),

B. Gains due to LLR-based Formulation of SCL Decoding

Our previous LL-based architecture of [19] and the LLR-based architecture with a radix-2L sorter presented in this paper are identical except that the former uses LLs while the latter uses LLRs. Therefore, by comparing these two

TABLE I
COMPARISON WITH LL-BASED IMPLEMENTATION

| | LL-Based [19] | | | LLR-Based | | |
|-------------------------|---------------|---------|---------|-----------|---------|---------|
| | $L = 2$ | $L = 4$ | $L = 8$ | $L = 2$ | $L = 4$ | $L = 8$ |
| Freq. (MHz) | 794 | 730 | 408 | 847 | 758 | 415 |
| Lat. (Cyc./bit) | 2.53 | 2.53 | 2.53 | 2.53 | 2.53 | 2.53 |
| T/P (Mbps) | 314 | 288 | 161 | 335 | 299 | 164 |
| Area (mm ²) | 1.38 | 2.62 | 5.38 | 0.88 | 1.75 | 3.87 |
| Efficiency | 227 | 110 | 30 | 380 | 171 | 42 |

TABLE II
CELL AREA BREAKDOWN FOR THE LL-BASED AND THE RADIX-2L
LLR-BASED SCL DECODERS ($R = \frac{1}{2}$, $N = 1024$)

| List Size | LL-Based [19] | LLR-Based | Reduction |
|----------------------------------|------------------------|------------------------|-----------|
| | $L = 2$ | | |
| Total Area (mm ²) | 1.38 | 0.88 | 36% |
| Memory (mm ²) | 1.07 | 0.80 | 25% |
| MCU (mm ²) | 0.28 | 0.06 | 79% |
| Metric Sorter (mm ²) | 1.34×10^{-3} | 0.75×10^{-3} | 44% |
| Other (mm ²) | 0.03 | 0.02 | 50% |
| List Size | $L = 4$ | | |
| | $L = 4$ | | |
| Total Area (mm ²) | 2.62 | 1.75 | 33% |
| Memory (mm ²) | 1.92 | 1.57 | 18% |
| MCU (mm ²) | 0.54 | 0.11 | 80% |
| Metric Sorter (mm ²) | 13.92×10^{-3} | 9.23×10^{-3} | 33% |
| Other (mm ²) | 0.15 | 0.06 | 60% |
| List Size | $L = 8$ | | |
| | $L = 8$ | | |
| Total Area (mm ²) | 5.38 | 3.87 | 28% |
| Memory (mm ²) | 4.08 | 3.46 | 15% |
| MCU (mm ²) | 0.82 | 0.18 | 78% |
| Metric Sorter (mm ²) | 70.65×10^{-3} | 54.05×10^{-3} | 24% |
| Other (mm ²) | 0.41 | 0.18 | 56% |

architectures we can specifically identify the improvements in terms of area and decoding throughput that arise directly from the reformulation of SCL decoding in the LLR domain.

The cycle count for our SCL decoder using the radix-2L sorter when decoding a $(1024, 512)$ polar code is $D_{SCL}(N, P, A) = 2592$ cycles (see (16) and (18)).

From Table I, we see that our LLR-based SCL decoder occupies 36%, 33%, and 28% smaller area than our LL-based SCL decoder of [19] for $L = 2$, $L = 4$, and $L = 8$, respectively. We present the area breakdown of the LL-based and the LLR-based decoders in Table II in order to identify where the area reduction mainly comes from and why the relative reduction in area decreases with increasing list size L . The memory area corresponds to the combined area of the LLR (or LL) memory, the partial sum memory, and the path memory. We observe that, in absolute terms, the most significant savings in terms of area come from the memory, where the area is reduced by up to 0.62 mm^2 for $L = 8$. On the other hand, in relative terms, the biggest savings in terms of area come from the MCU with an average area reduction of 79%. The relative reduction in the memory area decreases with increasing list size L . This happens because each bit-cell of the partial sum memory and the path memory contains L -to- L crossbars, whose size grows quadratically with L , while the LL (and LLR) memory grows only linearly in size with L . Thus, the size of the partial sum memory and the path memory, which are not affected by the LLR-based reformulation, becomes more significant as the list size is increased, and the relative reduction due to the LLR-based

TABLE III
RADIX-2L VS. PRUNED RADIX-2L SORTER

| | Radix-2L Sorter | | | Pruned Radix-2L Sorter | | |
|-------------------------|-----------------|---------|---------|------------------------|---------|---------|
| | $L = 2$ | $L = 4$ | $L = 8$ | $L = 2$ | $L = 4$ | $L = 8$ |
| Freq. (MHz) | 847 | 758 | 415 | 848 | 794 | 637 |
| Lat. (Cyc./bit) | 2.53 | 2.53 | 2.53 | 2.59 | 2.59 | 2.59 |
| T/P (Mbps) | 335 | 299 | 164 | 328 | 307 | 246 |
| Area (mm ²) | 0.88 | 1.75 | 3.87 | 0.9 | 1.78 | 3.85 |
| Efficiency | 380 | 171 | 42 | 364 | 172 | 64 |

formulation is decreased. Similarly, the relative reduction in the metric sorter area decreases with increasing L , because the LLR-based formulation only decreases the bit-width of the $L(2L - 1)$ comparators of the radix-2L sorter but it does not affect the size of the sorting logic, which dominates the sorter area as the list size is increased.

From Table I, we observe that the operating frequency (and, hence, the throughput) of our LLR-based decoder is 7%, 3%, and 2% higher than that of our LL-based SCL decoder of [19] for $L = 2$, $L = 4$, and $L = 8$, respectively.

Due to the aforementioned improvements in area and decoding throughput, the LLR-based reformulation of SCL decoding leads to hardware decoders with 67%, 55%, and 40% better hardware efficiency than the corresponding LL-based decoders of [19], for $L = 2$, $L = 4$, and $L = 8$, respectively.

C. Radix-2L Sorter versus Pruned Radix-2L Sorter

One may expect the pruned radix-2L sorter to always outperform the radix-2L sorter. However, the decoder equipped with the pruned radix-2L sorter needs to stall slightly more often to perform the additional sorting steps after groups of frozen bits. In particular, a (1024, 512) polar code contains $F_C(\mathcal{A}) = 57$ groups of frozen bits. Therefore, the total sorting latency for the pruned radix-2L sorter is $D_{MS}(\mathcal{A}) = |\mathcal{A}| + F_C(\mathcal{A}) = 569$ cycles (see (19)). Thus, we have $D_{SCL}(N, P, \mathcal{A}) = 2649$ cycles, which is an increase of approximately 2% compared to the decoder equipped with a full radix-2L sorter. Therefore, if using the pruned radix-2L does not lead to a more than 2% higher clock frequency, the decoding throughput will actually be reduced.

As can be observed in Table III, this is exactly the case for $L = 2$, where the LLR-based SCL decoder with the pruned radix-2L sorter has a 2% *lower* throughput than the LLR-based SCL decoder with the full radix-2L sorter. However, for $L \geq 4$ the metric sorter starts to lie on the critical path of the decoder and therefore using the pruned radix-2L sorter results in a significant increase in throughput of up to 50% for $L = 8$.

To provide more insight into the effect of the metric sorter on our SCL decoder, in Table IV we present the metric sorter delay and the critical path start- and endpoints of each decoder of Table III. The critical paths for $L = 2$ and $L = 4, 8$, are also annotated in Figure 2 with green dashed lines and red dotted lines, respectively. We denote the register of the controller which stores the internal LLR memory read address by R_{IM} . Moreover, let $D_{\hat{U}_s}$ and D_M denote a register of the partial sum memory and the metric memory, respectively. From Table IV, we observe that, for $L = 2$, the radix-2L sorter does not lie

TABLE IV
METRIC SORTER DELAY AND CRITICAL PATH START- AND ENDPOINTS FOR OUR LLR-BASED SCL DECODER USING THE RADIX-2L AND THE PRUNED RADIX-2L SORTERS.

| | Radix-2L Sorter | | | Pruned Radix-2L Sorter | | |
|---------------|-------------------|-----------------|-----------------|------------------------|----------|-----------------|
| | $L = 2$ | $L = 4$ | $L = 8$ | $L = 2$ | $L = 4$ | $L = 8$ |
| Delay (ns) | 0.50 ^a | 0.80 | 1.83 | 0.50 ^a | 0.54 | 1.09 |
| CP Startpoint | R_{IM} | D_M | D_M | R_{IM} | R_{IM} | D_M |
| CP Endpoint | D_M | $D_{\hat{U}_s}$ | $D_{\hat{U}_s}$ | D_M | D_M | $D_{\hat{U}_s}$ |

^a Note that the true delay of the pruned radix-2L sorter is always smaller than the delay of the radix-2L sorter. However, for $L = 2$, both sorters meet the synthesis timing constraint, which was set to 0.50 ns.

on the critical path of the decoder, which explains why using the pruned radix-2L sorter does not improve the operating frequency of the decoder. For $L \geq 4$ the metric sorter does lie on the critical path of the decoder and using the pruned radix-2L sorter results in a significant increase in the operating frequency of up to 53%. It is interesting to note that using the pruned radix-2L sorter eliminates the metric sorter completely from the critical path of the decoder for $L = 4$. For $L = 8$, even the pruned radix-2L sorter lies on the critical path of the decoder, but the delay through the sorter is reduced by 40%.

D. Comparison with LL-based SCL Decoders

In Table V, we compare our LLR-based decoder with the LL-based decoders of [23] and [22] along with our LL-based decoder of [19]. For the comparisons, we pick our SCL decoder with the best hardware efficiency for each list size, i.e., for $L = 2$ we pick the SCL decoder with the radix-2L sorter, while for $L = 4, 8$, we pick the SCL decoder with the pruned radix-2L sorter. Moreover, we pick the decoders with the best hardware efficiency from [22], i.e., the 4b-rSCL decoders.

1) *Comparison with [23]*: From Table V we observe that our LLR-based SCL decoder has an approximately 28% smaller area than the LL-based SCL decoder of [23] for all list sizes. Moreover, the throughput of our LLR-based SCL decoder is up to 70% higher than the throughput achieved by the LL-based SCL decoder of [23], leading to a 137%, 118%, and 120% better hardware efficiency for $L = 2$, $L = 4$ and $L = 8$, respectively.

2) *Comparison with [22]*: The synthesis results of [22] are given for a 65nm technology, which makes a fair comparison difficult. Nevertheless, in order to enable as fair a comparison as possible, we scale the area and the frequency to a 90nm technology in Table V (we have also included the original results for completeness). Moreover, the authors of [22] only provide synthesis results for $L = 2$ and $L = 4$. In terms of area, we observe that our decoder is approximately 57% smaller than the decoder of [22] for all list sizes. We also observe that for $L = 2$ our decoder has a 7% lower throughput than the decoder of [22], but for $L = 4$ the throughput of our decoder is 6% higher than that of [22]. Overall, the hardware efficiency of our LLR-based SCL decoder is 115% and 142% better than that of [22] for $L = 2$ and $L = 4$ respectively.

TABLE V
SCL DECODER SYNTHESIS RESULTS ($R = \frac{1}{2}$, $N = 1024$)

| | LLR-Based | | | LL-Based [19] | | | LL-Based [23] ^a | | | LL-Based [22] ^b | | | |
|-------------------------|-----------|---------|---------|---------------|---------|---------|----------------------------|---------|---------|-----------------------------|---------|---------|---------|
| | $L = 2$ | $L = 4$ | $L = 8$ | $L = 2$ | $L = 4$ | $L = 8$ | $L = 2$ | $L = 4$ | $L = 8$ | $L = 2$ | $L = 4$ | $L = 2$ | $L = 4$ |
| Technology | TSMC 90nm | | | TSMC 90nm | | | TSMC 90nm | | | Scaled to 90nm ^c | | ST 65nm | |
| Freq. (MHz) | 847 | 794 | 637 | 794 | 730 | 408 | 507 | 492 | 462 | 361 | 289 | 500 | 400 |
| Lat. (Cycles/bit) | 2.53 | 2.59 | 2.59 | 2.53 | 2.53 | 2.53 | 2.53 | 2.53 | 3.03 | 1.00 | 1.00 | 1.00 | 1.00 |
| T/P (Mbps) | 335 | 307 | 246 | 314 | 288 | 161 | 200 | 194 | 153 | 362 | 290 | 501 | 401 |
| Area (mm ²) | 0.88 | 1.78 | 3.58 | 1.38 | 2.62 | 5.38 | 1.23 | 2.46 | 5.28 | 2.03 | 4.10 | 1.06 | 2.14 |
| Efficiency | 380 | 172 | 69 | 227 | 110 | 30 | 163 | 79 | 29 | 178 | 71 | 473 | 187 |

^a The synthesis results in [23] are provided with up to 16 PEs per path. The reported numbers in this table are the corresponding synthesis results using 64 PEs per path and are courtesy of the authors of [23].

^b The authors of [22] use 3 quantization bits for the channel LLs and a tree SC architecture, while [19], [23] use 4 quantization bits for the channel LLs and a semi-parallel architecture with $P = 64$ PEs per path.

^c We use the standard assumption that area scales as s^2 and frequency scales as $1/s$, where s is the feature size.

E. CRC-Aided SCL Decoder

As discussed in Section II-C, the performance of the SCL decoder can be significantly improved if it is assisted for its final choice by means of a CRC which rejects some incorrect codewords from the final set of L candidates. However, there is a trade-off between the length of the CRC and the performance gain. A longer CRC, rejects more incorrect codewords but, at the same time, it degrades the performance of the inner polar code by increasing its rate. Hence, the CRC improves the overall performance if the performance degradation of the inner polar code is compensated by rejecting the incorrect codewords in the final list.

1) *Choice of CRC*: We picked three different CRCs of lengths $r = 4$, $r = 8$ and $r = 16$ from [32] with generator polynomials:

$$g(x) = x^4 + x + 1, \quad (21a)$$

$$g(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1, \text{ and} \quad (21b)$$

$$g(x) = x^{16} + x^{15} + x^2 + 1, \quad (21c)$$

respectively and evaluated the empirical performance of the SCL decoders of list sizes of $L = 2$, $L = 4$, $L = 8$, aided by each of these three CRCs in the regime of $E_b/N_0 = 1.5$ dB to $E_b/N_0 = 4$ dB.

For $L = 2$ it turns out that the smallest CRC, represented by the generator polynomial in (21a), is the best choice. Using longer CRCs at $E_b/N_0 \leq 3$ dB, the performance degradation of the polar code is dominant, causing the CRC-aided SCL decoder to perform *worse* than the standard SCL decoder. Furthermore, at higher SNRs, longer CRCs do not lead to a significantly better performance than the CRC-4.

For $L = 4$, allocating $r = 8$ bits for the CRC of (21b) turns out to be the most beneficial option. CRC-4 and CRC-8 will lead to almost identical FER at $E_b/N_0 < 2.25$ dB while CRC-8 improves the FER significantly more than CRC-4 at higher SNRs. Furthermore, CRC-16 leads to the same performance as CRC-8 at high SNRs and worse performance than CRC-8 in low-SNR regime.

Finally, for $L = 8$ we observe that CRC-16 of (21c) is the best candidate among the three different CRCs in the sense that the performance of the CRC-aided SCL decoder which uses this CRC is significantly better than that of the decoders using CRC-4 or CRC-8 for $E_b/N_0 > 2.5$ dB, while all three

TABLE VI
THROUGHPUT REDUCTION IN CRC-AIDED SCL DECODERS

| | | $L = 2$ | $L = 4$ | $L = 8$ |
|---------------|---------------|---------|---------|---------|
| Freq. (MHz) | | 847 | 794 | 637 |
| SCLD | $ A $ | 512 | 512 | 512 |
| | $F_C(A)$ | 57 | 57 | 57 |
| | Lat. (Cycles) | 2592 | 2649 | 2649 |
| | T/P (Mbits/s) | 335 | 307 | 246 |
| CA-SCLD | $ A $ | 516 | 520 | 528 |
| | $F_C(A)$ | 55 | 54 | 52 |
| | Lat. (Cycles) | 2596 | 2654 | 2660 |
| | T/P (Mbits/s) | 334 | 306 | 245 |
| Reduction (%) | | 0.2 | 0.2 | 0.4 |

decoders have almost the same FER at lower SNRs (and they all perform better than a standard SCL decoder).

In Figure 6, we compare the FER of the SCL decoder with that of the CA-SCLD for list sizes of $L = 2$, $L = 4$ and $L = 8$, using the above-mentioned CRCs. We observe that the CRC-aided SCL decoders perform significantly better than the standard SCL decoders.

2) *Throughput Reduction*: Adding r bits of CRC increases the number of information bits by r , while reducing the number of groups of frozen channels by *at most* r . As a result, the sorting latency is generally increased, resulting in a decrease in the throughput of the decoder. In Table VI we have computed this decrease in the throughput for different decoders and we see that the CRC-aided SCL decoders have slightly (at most 0.4%) reduced throughput. For this table, we have picked the best decoder at each list size in terms of hardware efficiency from Table III.

3) *Effectiveness of CRC*: The area of the CRC unit for all synthesized decoders is in less than $1 \mu\text{m}^2$ for the employed TSMC 90 nm technology. Moreover, the CRC unit does not lie on the critical path of the decoder. Therefore, it does not affect the maximum achievable operating frequency. Thus the incorporation of a CRC unit is a highly effective method of improving the performance of an SCL decoder. For example, it is interesting to note that the CA-SCLD with $L = 2$ has a somewhat lower FER than the standard SCL decoder with $L = 8$ (in both floating-point and fixed-point versions) in the regime of $E_b/N_0 > 2.5$ dB. Therefore, if a FER in the range of 10^{-3} to 10^{-6} is required by the application, using a CA-SCLD with list size $L = 2$ is preferable to a standard SCL decoder with list size $L = 8$ as the former has more than five

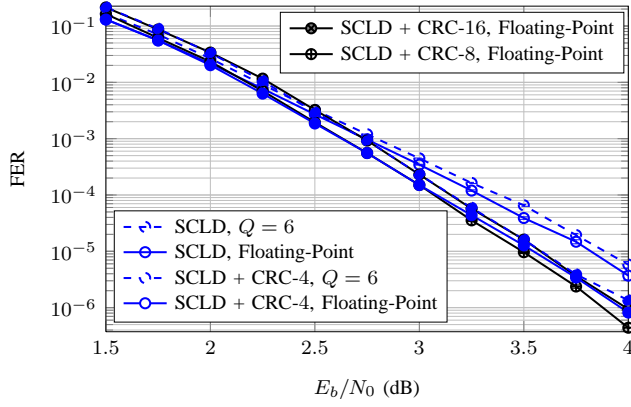
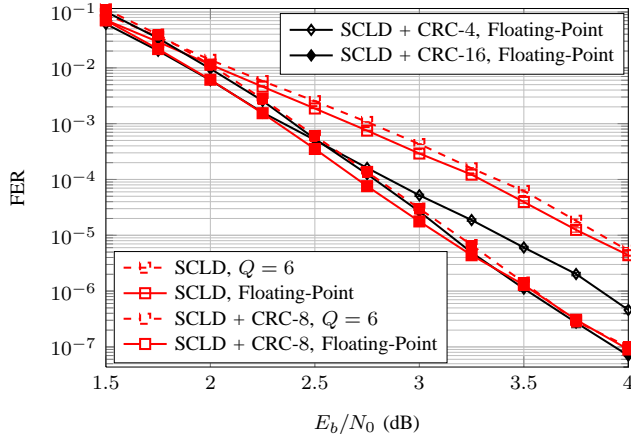
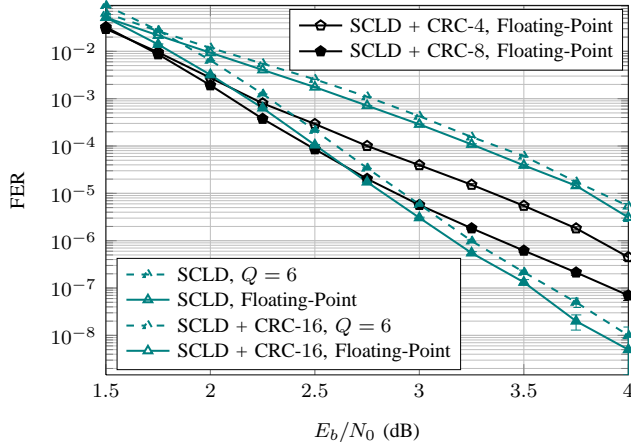
(a) $L = 2$ (b) $L = 4$ (c) $L = 8$

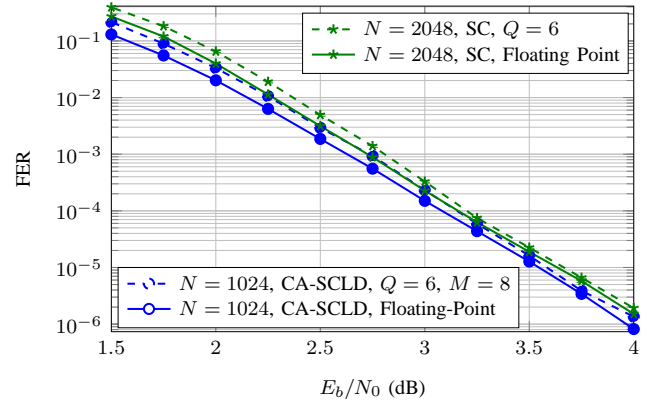
Fig. 6. The performance of LLR-based SCL decoders compared to that of CRC-Aided SCL decoders for $L = 2, 4, 8$. $M = 8$ quantization bits are used for the path metric in fixed-point simulations.

times higher hardware efficiency.

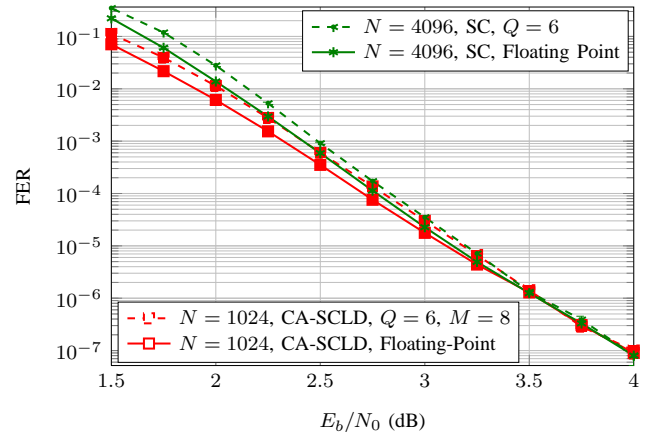
VII. DISCUSSION

A. SC Decoding or SCL Decoding?

Modern communication standards sometimes allow very long block-lengths to be used. The error-rate performance of polar codes under conventional SC decoding is significantly



(a) (2048, 1024) polar code under SC decoding versus (1024, 512) modified polar code under CA-SCLD with $L = 2$ and CRC-4 with generator polynomial (21a)



(b) (4096, 2048) polar code under SC decoding versus (1024, 512) modified polar code under CA-SCLD with $L = 4$ and CRC-8 with generator polynomial (21b)

Fig. 7. CA-SCLD with $L = 2, 4$, results in the same performance at block-length $N = 1024$ as the conventional SC decoding with $N = 2048$ and $N = 4096$, respectively.

improved if the block-length is increased. However, a long block-length implies long decoding latency and large decoders. Thus, an interesting question is whether it is better to use a long polar code with SC decoding or a shorter one with SCL decoding, for a given target block-error probability. In order to answer this question, we first need to find some pairs of short and long polar codes which have approximately the same block-error probability under SCL and SC decoding, respectively to carry out a fair comparison.

In Figure 7a we see that a (2048, 1024) polar code has almost the same block-error probability under SC decoding as a (1024, 512) modified polar code under CA-SCLD with list size $L = 2$ and CRC-4 of (21a). Similarly, in Figure 7b we see that a (4096, 2048) polar code has almost the same block-error probability under SC decoding as an (1024, 512) modified polar code decoded under CA-SCLD with list size $L = 4$ and CRC-8 of (21b).

As mentioned earlier, our SCL decoder architecture is based on the SC decoder of [5]. In Table VII we present the synthesis results for the SC decoder of [5] at block lengths $N = 2048$ and $N = 4096$ and compare them with that of our LLR-based

TABLE VII
LLR-BASED SC DECODER VS. SCL DECODER SYNTHESIS RESULTS

| | SC | CA-SCLD $L = 2$, CRC-4 | SC | CA-SCLD $L = 4$, CRC-8 |
|-------------------------|------|----------------------------|------|----------------------------|
| N | 2048 | 1024 | 4096 | 1024 |
| Freq. (MHz) | 870 | 847 | 806 | 794 |
| Lat. (Cyc./bit) | 2.05 | 2.54 | 2.06 | 2.59 |
| Lat. (Cyc.) | 4192 | 2596 | 8448 | 2654 |
| T/P (Mbps) | 425 | 334 | 391 | 306 |
| Area (mm ²) | 0.78 | 0.88 | 1.51 | 1.78 |

SCL decoder, when using the same TSMC 90nm technology and identical operating conditions. For all decoders, we use $P = 64$ PEs per path and $Q = 6$ bits for the quantization of the LLRs.

First, we see that the SCL decoders occupy an approximately 15% larger area than their SC decoder counterparts. This may seem surprising, as it can be verified that an SC decoder for a code of length LN requires more memory (LLR and partial sum) than the memory (LLR, partial sum, and path) required by an SCL decoder with list size L for a code of length N , and we know that the memory occupies the largest fraction of both decoders. This discrepancy is due to the fact that the copying mechanism for the partial sum memory and the path memory still uses $L \times L$ crossbars, which occupy significant area. It is an interesting open problem to develop an architecture that eliminates the need for these crossbars.

Moreover, we observe that both SC decoders can achieve a slightly higher operating frequency than their corresponding SCL decoders, although the difference is less than 3%. However, the per-bit latency of the SC decoders is about 20% smaller than that of the SCL decoders, due to the sorting step involved in SCL decoding. The smaller per-bit latency of the SC decoders combined with their slightly higher operating frequency, make the SC decoders have an almost 27% higher throughput than their corresponding SCL decoders.

However, from Table VII we see that the SCL decoders have a significantly lower per-codeword latency. More specifically, the SCL decoder with $N = 1024$ and $L = 2$ has a 38% lower per-codeword latency than the SC decoder with $N = 2048$, and the SCL decoder with $N = 1024$ and $L = 4$ has a 68% lower per-codeword latency than the SC decoder with $N = 4096$. Thus, for a fixed FER, our LLR-based SCL decoders provide a solution of reducing the per-codeword latency at a small cost in terms of area, rendering them more suitable for low-latency applications than their corresponding SC decoders.

B. Simplified SC and SCL Decoders

There has been significant work done to reduce the latency of SC decoders [10]–[13] by pruning the decoding graph, resulting in *simplified SC* (SSC) decoders. The SC decoder architecture of [5], used in our comparison above, does not employ any of these techniques. Since our SCL decoder uses L SC decoders, it seems evident that any architectural and algorithmic improvements made to the SC decoder itself will be beneficial to the LLR-based SCL decoder as well. However, the family of SSC decoders does not seem to be directly applicable to our LLR-based SCL decoder. This

happens because, in order to keep the path metric updated, we need to calculate the LLRs even for the frozen bits. As discussed in Section III, it is exactly these LLRs that lead to the improved performance of the SCL decoder with respect to the SC decoder. However, alternative and promising pruning approaches which have been recently introduced in the context of LL-based SCL decoding [22], [33], are fully applicable to LLR-based SCL decoding.

VIII. CONCLUSION

In this work, we introduced an LLR-based path metric for SCL decoding of polar codes, which enables the implementation of a numerically stable LLR-based SCL decoder. Moreover, we showed that we can simplify the sorting task of the SCL decoder by using a pruned radix- $2L$ sorter which exploits the properties of the LLR-based path metric. The LLR-based path metric is not specific to SCL decoding and can be applied to any other tree-search based decoder (e.g., stack SC decoding [34]).

We implemented a hardware architecture for an LLR-based SCL decoder and we presented synthesis results for various list sizes. Our synthesis results clearly show that our LLR-based SCL decoder has a significantly higher throughput *and* lower area than all existing decoders in the literature, leading to a substantial increase in hardware efficiency of up to 137%.

Finally, we showed that adding the CRC unit to the decoder and using CA-SCLD is an easy way of increasing the hardware efficiency of our SCL decoder at a given block-error probability as the list size can be decreased. Specifically, our CA-SCLD at list size $L = 2$ has somewhat lower block-error probability *and* more than five times better hardware efficiency than our standard SCLD at list size $L = 8$.

REFERENCES

- [1] E. Arkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.
- [2] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *Proceedings of 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1665–1668.
- [3] A. J. Raymond and W. J. Gross, "Scalable successive-cancellation hardware decoder for polar codes," in *Proceedings of 2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec. 2013, pp. 1282–1285.
- [4] A. Pamuk and E. Arkan, "A two phase successive cancellation decoder architecture for polar codes," in *Proceedings of 2013 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2013, pp. 957–961.
- [5] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, Jan. 2013.
- [6] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2429–2441, Mar. 2013.
- [7] G. Sarkis, P. Giard, A. Vardy, C. Thibault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, May 2014.
- [8] Y. Fan and C.-Y. Tsui, "An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3165–3179, Jun. 2014.
- [9] A. Mishra, A. J. Raymond, L. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. J. Gross, "A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS," in *Proceedings of 2012 IEEE Asian Solid State Circuits Conference (A-SSCC)*, Nov. 2012, pp. 205–208.

- [10] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, Oct. 2011.
- [11] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency SC polar decoder architectures," in *Proceedings of 2012 IEEE International Conference on Communications (ICC)*, Jun. 2012, pp. 3471–3475.
- [12] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Communications Letters*, vol. 17, no. 4, pp. 725–728, Apr. 2013.
- [13] C. Zhang and K. K. Parhi, "Latency analysis and architecture design of simplified SC polar decoders," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, vol. 61, no. 2, pp. 115–119, Feb. 2014.
- [14] E. Arikan and E. Telatar, "On the rate of channel polarization," in *Proceedings of 2009 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2009, pp. 1493–1495.
- [15] I. Tal and A. Vardy, "List decoding of polar codes," in *Proceedings of 2011 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2011, pp. 1–5.
- [16] —, "List decoding of polar codes," *arXiv e-prints*, vol. abs/1206.0050, 2012. [Online]. Available: <http://arxiv.org/abs/1206.0050>
- [17] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, Oct. 2012.
- [18] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Communications Letters*, vol. 16, no. 12, pp. 2044–2047, Dec. 2012.
- [19] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, vol. 61, no. 8, pp. 609–613, May 2014.
- [20] J. Lin and Z. Yan, "Efficient list decoder architecture for polar codes," in *Proceedings of 2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 1022–1025.
- [21] C. Zhang, X. You, and J. Sha, "Hardware architecture for list successive cancellation polar decoder," in *Proceedings of 2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 209–212.
- [22] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (to appear)*, 2014. [Online]. Available: <http://dx.doi.org/10.1109/TVLSI.2014.2359793>
- [23] J. Lin and Z. Yan, "An efficient list decoder architecture for polar codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (to appear)*, 2015. [Online]. Available: <http://dx.doi.org/10.1109/TVLSI.2014.2378992>
- [24] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," in *Proceedings of 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 3903–3907.
- [25] G. Sarkis, P. Giard, A. Vardy, C. Thibault, and W. J. Gross, "Increasing the speed of polar list decoders," in *Proceedings of 2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2014, pp. 1–6.
- [26] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," in *Proceedings of 2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2014, pp. 1–6.
- [27] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.
- [28] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, ser. North-Holland Mathematical Library. North-Holland, 1978.
- [29] "IEEE standard for air interface for broadband wireless access systems," *IEEE Std 802.16TM-2012*, Aug. 2012.
- [30] I. Tal, "Private communication," Aug. 2014.
- [31] L. G. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2342–2346, Dec. 2012.
- [32] Wikipedia. (2014, Sep.) Polynomial representations of cyclic redundancy checks — wikipedia, the free encyclopedia. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Polynomial_representations_of_cyclic_redundancy_checks
- [33] C. Xiong, J. Lin, and Z. Yan, "Symbol-decision successive cancellation list decoder for polar codes," *arXiv e-prints*, vol. abs/1501.04705, Jan. 2015. [Online]. Available: <http://arxiv.org/abs/1501.04705>
- [34] K. Chen, K. Niu, and J. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Transactions on Communications*, vol. 61, no. 8, pp. 3100–3107, Aug. 2013.